Ontologies with OWL

Lecture 4

OWL Ontologies

- Ontologies are used to capture knowledge about some domain of interest.
- An ontology describes the **concepts** in the domain and also the **relationships** that hold between those concepts.
- Different ontology languages provide different facilities.
- **Web Ontology Language (OWL)** is a family of knowledge representation languages.

OWL Ontologies

- Since 2004, OWL has been a World Wide Web Consortium (W3C) recommended standard for ontology modelling.
- OWL has experienced a significant increase in popularity across various application domains.

OWL Ontologies

- OWL makes it possible to describe concepts in an unambiguous manner based on set theory and logic.
- Complex concepts can be built up out of simpler concepts.
- Logical model allows the use of a reasoner which can check whether all of the statements and definitions in the ontology are mutually consistent and can also recognize which concepts fit under which definitions.

Species of OWL

- OWL offers three sublanguages (species) to accommodate varying expressivity needs:
  1. **OWL Lite** – least expressive
  2. **OWL DL** – includes OWL Lite
  3. **OWL Full** – includes both OWL DL and OWL Lite

Components of OWL Ontologies

- An OWL ontology consists of
  1. Individuals
  2. Properties
  3. Classes
- OWL ontologies are an implementation of **Description Logic (DL)** which is a decidable subset of First Order Logic.
- A class in OWL is a **set**, a property is a **binary relation**, and an individual is an **element of a set**.

Components of OWL Ontologies

- Other concepts from set theory are also implemented in OWL such as **Disjoint sets**, the **Empty set** (owl:Nothing), **inverse relations, transitive relations**, and many more.
- An understanding of the basic concepts of set theory will help the user get the most out of OWL but is not required.

Individuals

- Individuals, represent **objects** in the **domain of interest.**
- An important difference between OWL and most programming and knowledge representation languages is that OWL **does not** use the **Unique Name Assumption (UNA).**
- This means that two different names could actually refer to the same individual.

Individuals

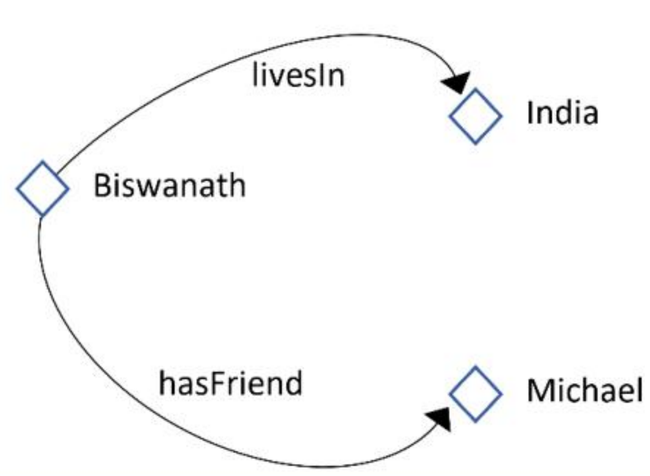
- Example
  - "Queen Elizabeth", "The Queen" and "Elizabeth Windsor" might all refer to the same individual.
- In OWL, it must be **explicitly stated** that individuals are the same as each other, or different from each other.

Individuals

- Individuals are also known as **instances**.
- Individuals can be referred to as being "**instances of classes**".
- For example, figure shows a representation of some individuals in some domain.

Properties

- **Properties** are binary relations on individuals.
- That is, properties link two individuals together.
- E.g., property ***hasFriend*** might link the individual Biswanath to the individual Michael.



Properties

- Properties can have **inverses**.
  - E.g.- Inverse of ***hasChild*** is ***hasparent***.
- Properties can be limited to having a **single value**.
- These single value properties are known as **functional properties.**
- They can also be either **transitive** or **symmetric**.

Classes

- OWL classes are **sets that contain individuals.**
- They are described using formal (mathematical) descriptions that state precisely the requirements for membership of the class.
- Example:
  - Class *Cat* would contain all the individuals that are cats in our domain of interest.

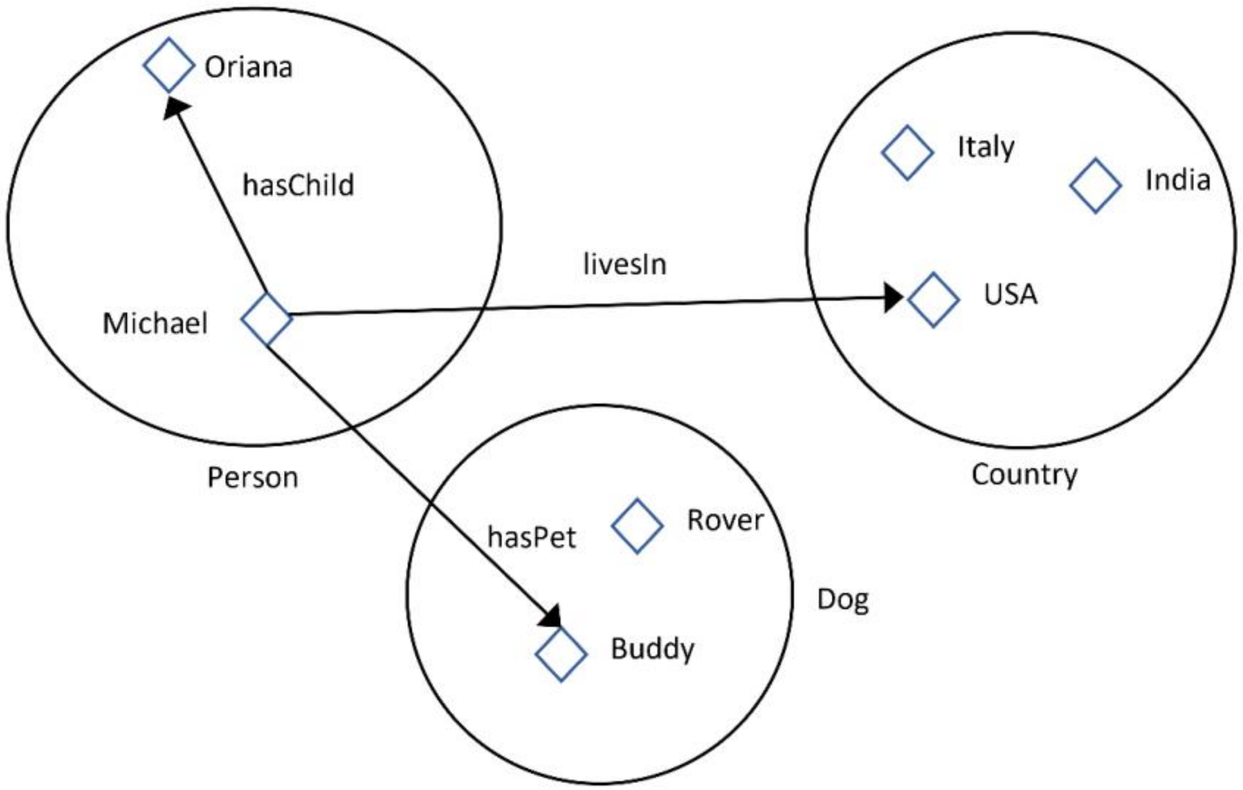
Classes

- Classes may be organized into a **superclass-subclass hierarchy**, which is also known as a **taxonomy**.
- However, taxonomies are often **trees**. I.e., each node has only one parent node.
- Class hierarchies in OWL are not restricted to be trees and multiple inheritance can be a powerful tool to represent data in an intuitive manner.

Classes

- **Subclasses** specialize (aka are subsumed by) their **superclasses**.
- For example, consider the classes *Animal* and *Dog*.
  - *Dog* might be a subclass of *Animal* (so *Animal* is the superclass of *Dog*).
  - This says that All dogs are animals, All members of the class
  - Dog are members of the class *Animal*.

Classes


Oriana
hasChild
Michael
livesIn
Person
Italy
India
USA
Country
hasPet
Rover
Buddy
Dog

16

Classes

- OWL and Protégé provide a language that is called Description Logic or DL for short.
- One of the key features of DL is that these superclass-subclass relationships (aka subsumption relationships) can be computed automatically by a reasoner.
- In OWL classes can be built up of descriptions that specify the conditions that must be satisfied by an individual for it to be a member of the class.

Building Ontologies with Protégé

- Protégé is a free, open-source ontology editor and framework for building intelligent systems.

- One of the benefits of Protégé is that it presents an intuitive GUI that enables domain experts to define models without a background in set theory.

http://protege.stanford.edu

Tutorial

A Practical Guide to Building OWL Ontologies Using Protégé 5.5 and Plugins

Edition 3.2

8 October 2021

Michael DeBellis

https://tinyurl.com/NewPizzaTutorialV3-2

Named Classes

- Main building blocks of an OWL ontology are **classes**.
- All empty ontologies contains one class called ***owl:Thing***.
- OWL classes are sets of individuals.
- Class ***owl:Thing*** is the class that represents the set containing all individuals.
- Because of this all classes are subclasses of ***owl:Thing***.

OWL Naming Conventions

- There are no mandatory naming conventions for OWL entities.
- A best practice is to select one set of naming conventions and then abide by that convention across your organization.
- For this lecture let's define the following standard.

OWL Naming Conventions

- Class names should start with a capital letter and should not contain spaces (CamelBack notation).
  - E.g. - Pizza, PizzaTopping, MargheritaPizza
- Class names are always singular rather than plural.
  - E.g., Pizza rather than Pizzas, PizzaTopping rather than PizzaToppings.

Disjoint Classes

- If classes are **disjoint**, **no individual (or object) cannot be an instance of more than one of the sibling classes**.
- OWL classes are assumed to 'overlap', i.e., by **default they are not disjoint**.

Disjoint Classes

- If In order to 'separate' a group of classes, we **must explicitly declare them as disjoint** from one another.
- This ensures that an individual which has been asserted to be a **member of one of the classes in the group cannot be a member of any other classes in that group**.

OWL Properties

- OWL properties **represent relationships**.
- There are 3 main types of properties;
  1. **Object** properties
  2. **Datatype** properties
  3. **Annotation** properties

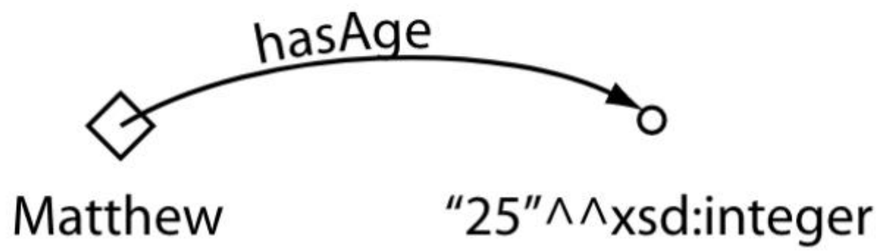
OWL Properties

- Object properties are **relationships between two individuals**.
- Object properties **link an individual to an individual**.

An object property linking the individual Matthew to the individual Gemma

OWL Properties

- **Datatype properties** describe relationships between individuals and data values.



A datatype property linking the individual Matthew to the data literal '25', which has a type of an xsd:integer.

OWL Properties

- **Annotation properties** can be used to add information (metadata — data about data) to classes, individuals and object/datatype properties.

An annotation property, linking the class 'JetEngine' to the data literal (string) ''Matthew Horridge".

OWL Properties

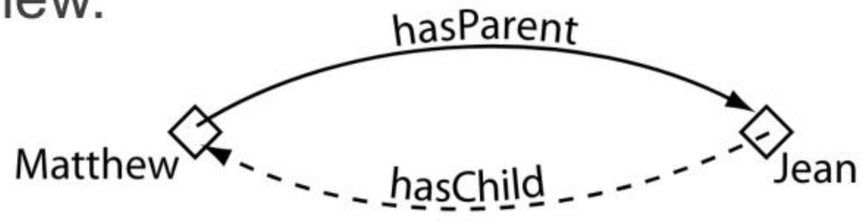
- There is no strict naming convention for properties.
- It is recommend that property names start with a lower case letter, have no spaces and have the remaining words capitalised.
- Also recommend that properties are **prefixed with the word 'has', or the word 'is'**.
- E.g.- *hasPart, isPartOf, hasManufacturer, isProducerOf*

OWL Properties

- In OWL, **properties may have sub-properties.**
- So that it is possible to form hierarchies of properties.
- Sub-properties specialise their super properties.
  - E.g.- Property *hasMother* might specialise the more general property of *hasParent*.
- All OWL properties are ultimately a sub-property of ***owl:topObjectProperty***.

Inverse Properties

- Each object property may have a corresponding **inverse property.**
- If some property links individual a to individual b then its **inverse** property will link individual b to individual a.
  - E.g.- If Matthew *hasParent* Jean, then because of the inverse property we can infer that Jean *hasChild* Matthew.

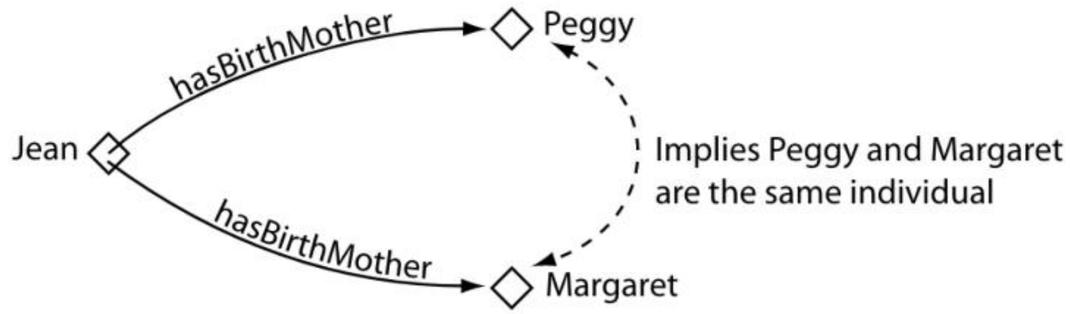


OWL Object Property Characteristics

- OWL allows the meaning of properties to be enriched through the use of property characteristics.
  1. Functional properties
  2. Inverse Functional properties
  3. Transitive properties
  4. Symmetric properties
  5. Asymmetric properties
  6. Reflexive properties
  7. Irreflexive properties

Functional Properties

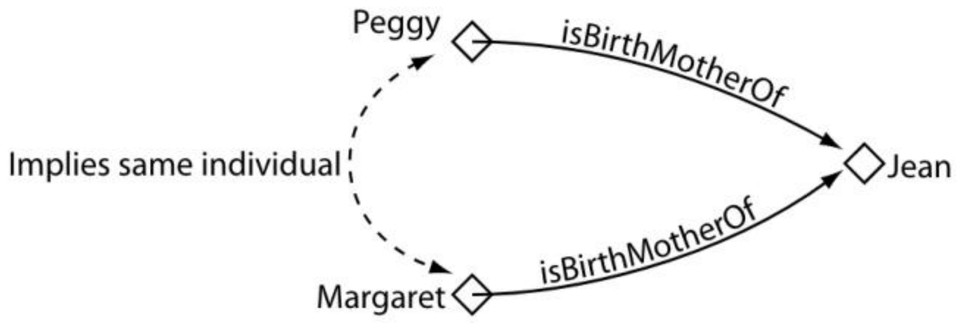
- If a property is **functional**, for a given individual, there can be at most one individual that is related to the individual via the property.



- Functional properties are also known as **single valued properties** and also **features**.

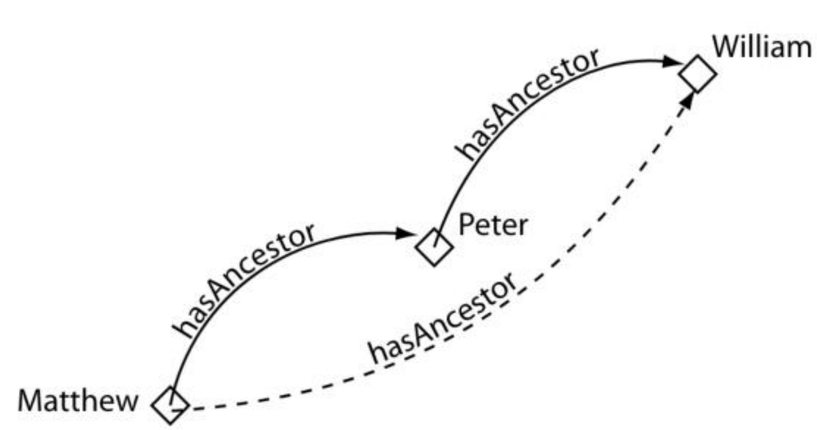
Inverse Functional Properties

- If a property is **inverse functional** then it means that the inverse property is functional.
- For a given individual, there can be at most one individual related to that individual via the property.



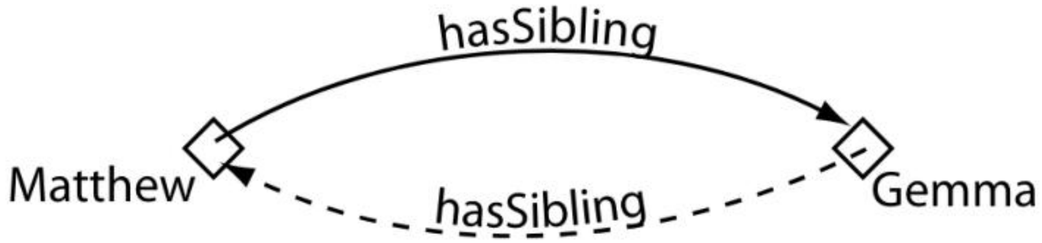
Transitive Properties

- If a property is **transitive**, and the property relates individual a to individual b, and also individual b to individual c, then we can infer that individual a is related to individual c via property P.



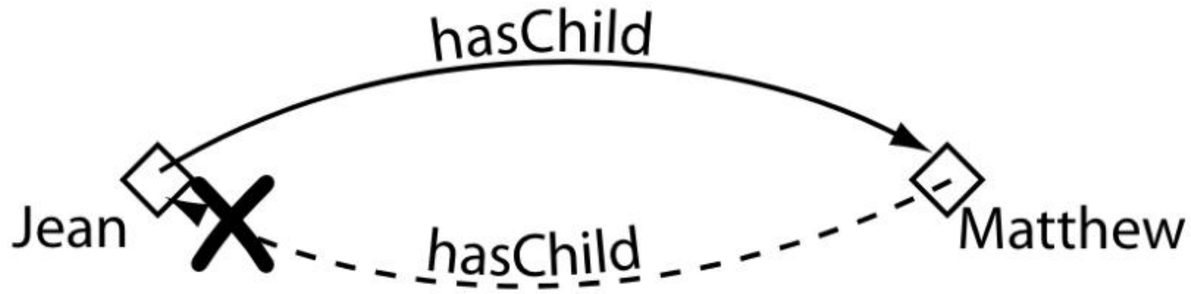
Symmetric Properties

- If a property P is **symmetric**, and the property relates individual a to individual b then individual b is also related to individual a via property P.
- Put another way, the property is its own inverse property.



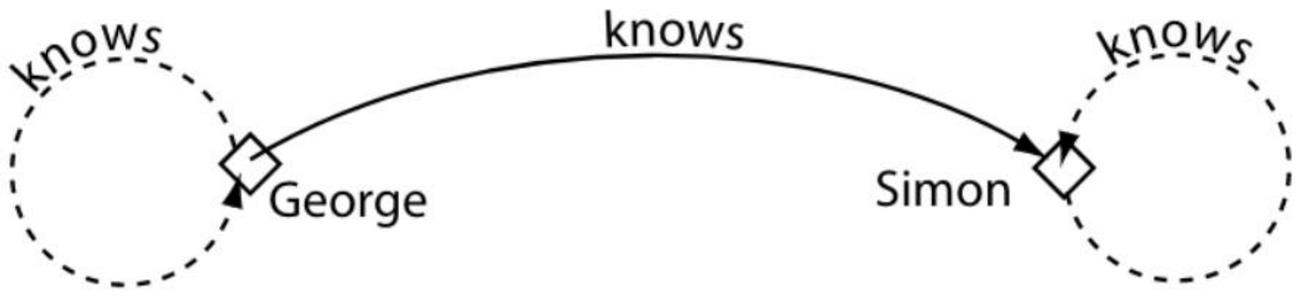
Asymmetric Properties

- If a property P is **asymmetric**, and the property relates individual a to individual b then individual b cannot be related to individual a via property P.



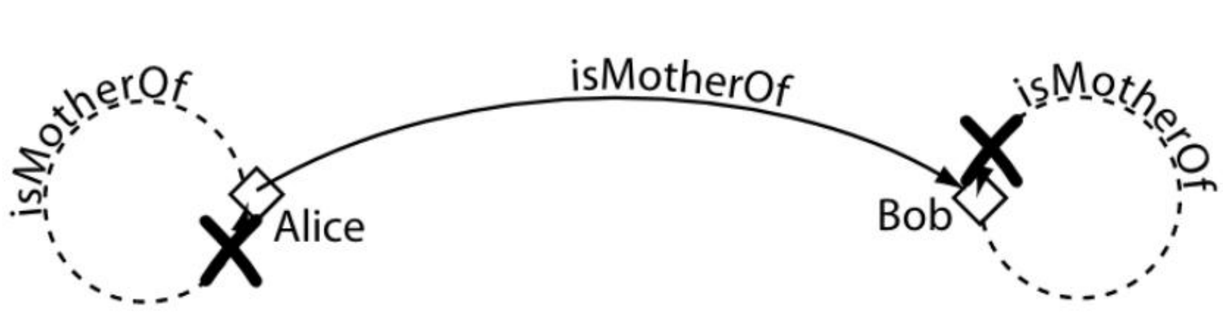
Reflexive Properties

- A property P is said to be **reflexive** when the property must relate individual a to itself.



Irreflexive Properties

- If a property P is **irreflexive**, it can be described as a property that relates an individual a to individual b, where individual a and individual b are not the same.



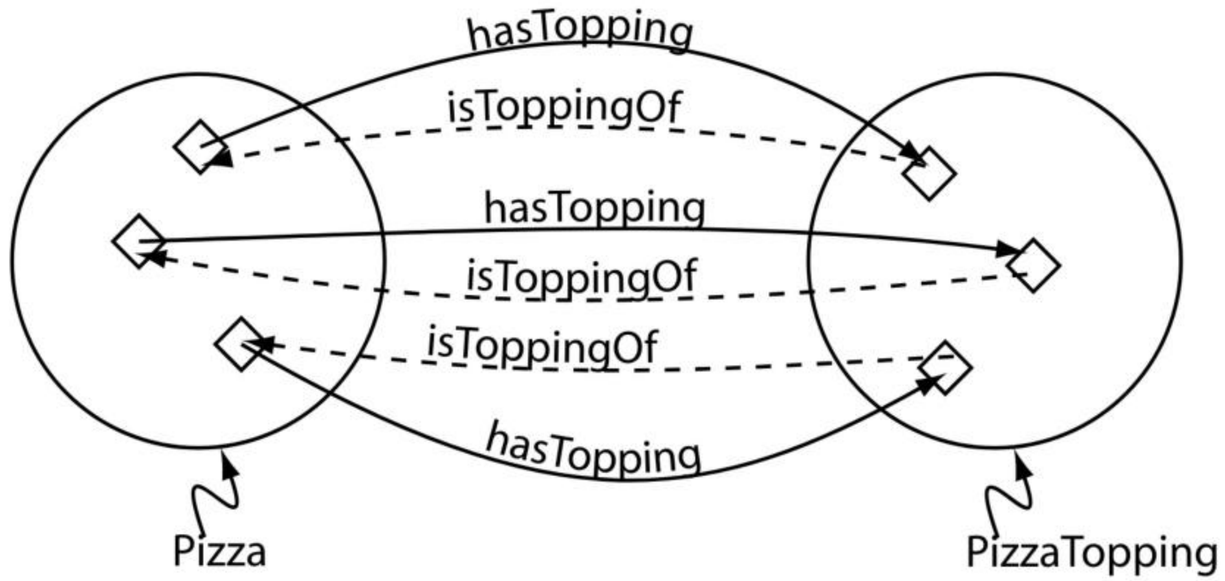
Property Domains and Ranges

- Properties may have a **domain** and a **range** specified.
- The **domain** of a property is the **set of all objects that can have that property asserted about it.**
- The **range** is the **set of all objects that can be the value of the property.**

Property Domains and Ranges

- In general, domain for a property is the range for its inverse, and the range for a property is the domain for its inverse.
- E.g.- hasTopping –> Pizza (Domain), PizzaTopping (Range).
- The domain and range for isToppingOf are the domain and range for hasTopping **swapped** over.

Property Domains and Ranges



Property Domains and Ranges

- It is possible to **specify multiple classes as the range for a property**.
- If multiple classes are specified in Protege the range of the property is interpreted to be the **intersection of the classes**.
- E.g, if the range of a property has the classes Man and Woman listed in the range view, the range of the property will be interpreted as Man intersection Woman.

Recall: OWL Properties

- There are two types of OWL properties for describing a domain: **Object properties and Data properties.**
- Object properties have **classes** as their **domain and range**.
- Data properties have **classes as their domain** and **simple datatypes as range** (such as xsd:string or xsd:dateTime as their).

Recall: OWL Properties

Example:

- Individual *Michael* is related to the individual USA by the property ***livesIn***.
- Consider all the individuals who are an instance of Person and also have the same relation, that each ***livesIn*** the USA.
- This group is a set or OWL class such as *USAResidents*.

Property Restrictions

- In OWL a class can be defined by describing the various **properties** and **values** that hold for **all individuals** in the class. Such definitions are called **restrictions** in OWL.

Example:

- Class of individuals with at least one *hasChild* relation.
- Class of individuals with 2 or more *hasChild* relations.

Property Restrictions

Example:

- Class of individuals that have at least one *hasTopping* relationship to individuals that are members of *MozzarellaTopping* – i.e. the class of things that have at least a mozzarella topping.
- Class of individuals that are Pizzas and only have *hasTopping* relations to instances of the class *VegetableTopping* (i.e., VegetarianPizza).

Property Restrictions

- In OWL we can describe all of the above classes using restrictions.
- OWL restrictions in OWL fall into **three** main categories:
  1. Quantifier Restrictions
  2. Cardinality Restrictions
  3. hasValue Restrictions

Property Restrictions

Quantifier Restrictions

- **Quantifier** restrictions describe that **a property must have some or all values that are of a particular class**.
- Quantifier restrictions can be further categorised into
  1. Existential Restrictions
  2. Universal Restrictions

Property Restrictions

Cardinality Restrictions

- These restrictions describe **the number of individuals that must be related to a class by a specific property**.

hasValue Restrictions

- These describe **specific values that a property must have.**

Property Restrictions

- A restriction always describes a class.
- Sometimes it can be a defined class or sometimes it may be an anonymous class.
- In all cases the class contains all of the individuals that satisfy the restriction.
- That is all of the individuals that have the relationships required to be a member of the class.

Existential Restrictions

- Existential restrictions are by far the most common type of restrictions in OWL ontologies.
- **An existential restrictions** describe a class of individuals that **participate in at least one (some) relationship along a specified property to individuals that are members of a specified class**.

Existential Restrictions

- Existential restrictions are also known as **Some Restrictions**, or as **some values** from restrictions.
- In OWL, the keyword **'some'** is used to denote existential restrictions.

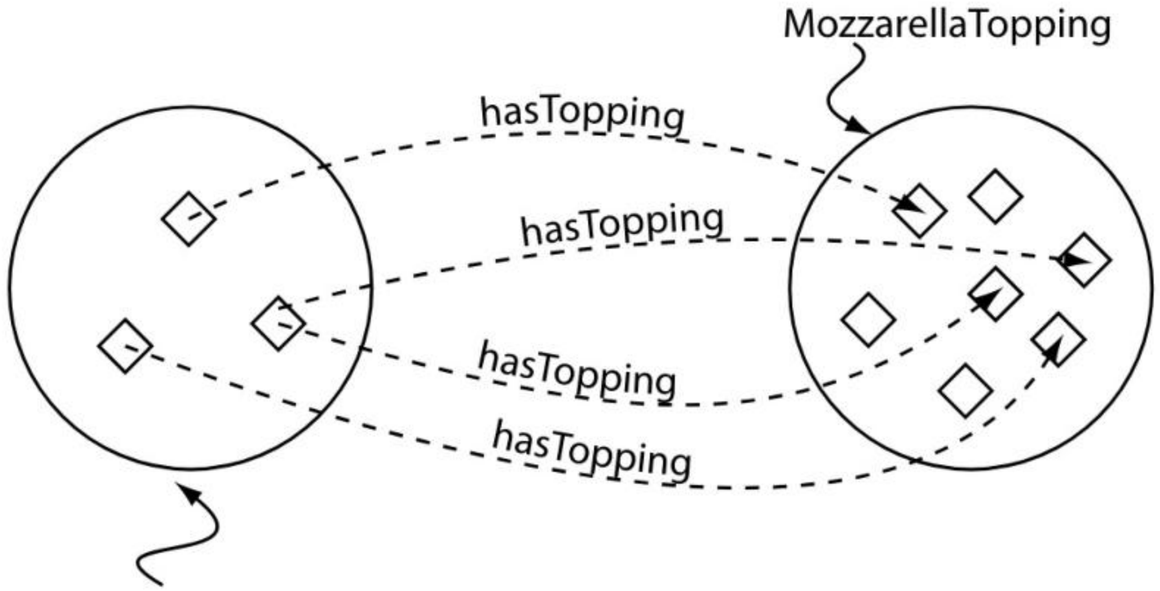
Example:

- Class of individuals who have at least one (or some) hasTopping relation to instances of VegetableTopping.

Existential Restrictions

- Example:
  - ***hasTopping some MozzarellaTopping*** is an existential restriction, which acts along the *hasTopping property*, and has a filler *MozzarellaTopping*.
  - This restriction describes the class of individuals that have at least one *hasTopping* relationship to an individual that is a member of the class *MozzarellaTopping*.

Existential Restrictions



Things that have at least one
MozzarellaTopping
(hasTopping some MozzarellaTopping)

Necessary & Sufficient Conditions

- All of the classes that were created so far **have only used necessary conditions** (axioms) to describe them.
- Necessary conditions (axioms) can be read as, "**If something is a member of this class then it is necessary to fulfil these conditions**".
- With necessary conditions (axioms) alone, it is not possible to say that: "**If something fulfils these conditions then it must be a member of this class**".

Necessary & Sufficient Conditions

Example

- If something is a *CheesyPizza* it is necessarily a Pizza and it is necessary for it to have at least one topping that is a kind of *CheeseTopping*.
- Suppose an individual
  - is a member of the class *Pizza*.
  - has at least one kind of *CheeseTopping*.

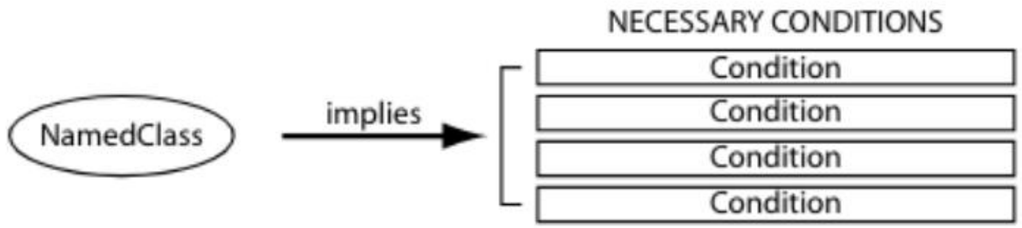
Necessary & Sufficient Conditions

- Current description of *CheesyPizza* this knowledge is not sufficient to determine that the individual is a member of the class *CheesyPizza*.
- To make this possible we need to change the conditions for *CheesyPizza* from necessary conditions to **necessary AND sufficient conditions**.

Necessary & Sufficient Conditions

- This means that not only are the conditions necessary for membership of the class CheesyPizza, they are also sufficient to determine that any random individual that satisfies them must be a member of the class CheesyPizza.

Necessary & Sufficient Conditions

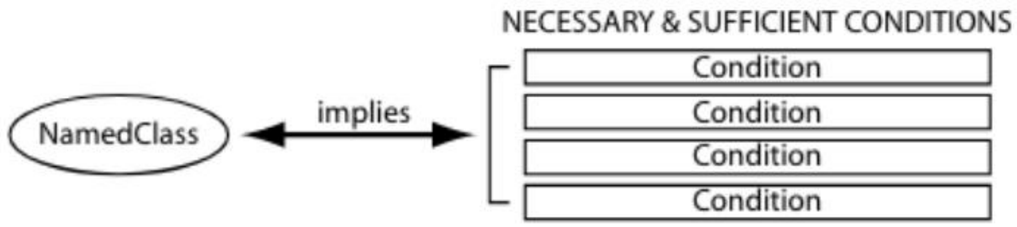


- If an individual is a member of 'NamedClass' then it must satisfy the conditions.
- However if some individual satisfies these necessary conditions, we cannot say that it is a member of 'Named Class' (the conditions are not 'sufficient' to be able to say this) - this is indicated by the **direction of the arrow**.

Necessary & Sufficient Conditions

- If class A is described using necessary conditions, then we can say that if an individual is a member of class A it must satisfy the conditions.
- We cannot say that any (random) individual that satisfies these conditions must be a member of class A.

Necessary & Sufficient Conditions



- If an individual is a memeber of ‘NamedClass’ then it must satisfy the conditions.
- If some individual satisfies the condtions then the individual must be a member of ‘NamedClass’ - this is indicated by the **double arrow**.

Necessary & Sufficient Conditions

- However, if class A is now defined using **necessary and sufficient conditions**, we can say that if an individual is a member of the class A it must satisfy the conditions and we can now say that if any (random) individual satisfies these conditions then it must be a member of class A.
- Conditions are not only necessary for membership of A but also sufficient to determine that something satisfying these conditions is a member of A.

Describing & Defining Classes

- Previously defined properties to define some more interesting classes.
- There are 3 types of classes in OWL:
  1. Primitive classes
  2. Defined classes
  3. Anonymous classes

Describing & Defining Classes

Primitive Classes

- These are classes that are defined by conditions that are necessary (but not sufficient) to hold for any individuals that are instances of that class or its subclasses.
- Condition may be as simple as:

  Class A is a subclass of class B.

Describing & Defining Classes

Defined Classes

- These are classes that are defined by both necessary and sufficient conditions.
- When the reasoner encounters an individual that satisfies all the conditions for a defined class it will make the inference that the individual is an instance of that class.

Describing & Defining Classes

Anonymous Classes

- They are created by the reasoner when you use class expressions.
- For example, if you define the range of a property to be *PizzaTopping* or *PizzaBase* then the reasoner will create an anonymous class representing the intersection of those two classes.

Primitive and Defined Classes

- A class that **only has necessary conditions** is called a **primitive** class.
- A class that has **necessary and sufficient conditions** is known as a **defined** class.
- Necessary conditions are simply called **Superclasses** in Protege.
- Necessary and sufficient condition are called **Equivalent** classes.

Universal Restrictions

- Existential restrictions specify the existence of at least one relationship along a given property to an individual that is a member of a specific class.
- Existential restrictions **do not mandate** that the only relationships for the given property that can exist must be to individuals that are members of the specified filler class.

Universal Restrictions

E.g. :hasTopping some MozzarellaTopping

- This is an existential restriction that describe the individuals that have at least one relationship along the property hasTopping to an individual that is a member of the class MozzarellaTopping.
- Does not imply that all of the hasTopping relationships must be to a member of the class MozzarellaTopping.

Universal Restrictions

- **Universal restrictions** constrain the relationships along a given property to individuals that are members of a specific class.
- Universal restrictions correspond to the symbol ∀ in First Order Logic.
- In Protege 4 the keyword '**only**' is used.

Universal Restrictions

- Example:

  $\forall$ *hasTopping VegetableTopping*

- This universal restriction describes the individuals all of whose *hasTopping* relationships are to members of the class *VegetableTopping*
  - The individuals do not have a *hasTopping* relationship to individuals that aren't members of the class *VegetableTopping*.

Cardinality Restrictions

- In OWL we can describe the class of individuals that have at least, at most or exactly a specified number of relationships with other individuals or datatype values.
- The restrictions that describe these classes are known as **Cardinality Restrictions.**

Cardinality Restrictions

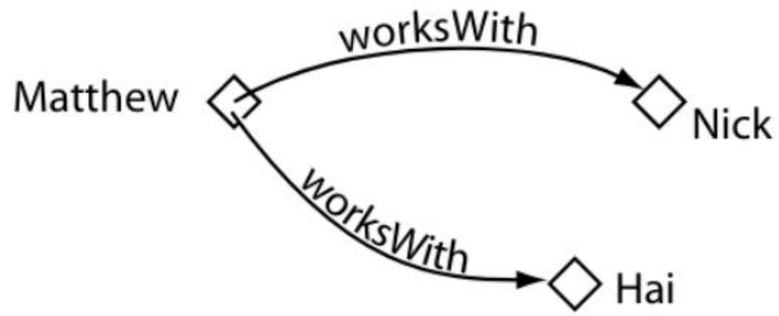
- For a given property P, a **Minimum Cardinality Restriction** specifies the minimum number of P relationships that an individual must participate in.
- A **Maximum Cardinality Restriction** specifies the maximum number of P relationships that an individual can participate in.

Cardinality Restrictions

- A Cardinality Restriction specifies the exact number of P relationships that an individual must participate in.
- Relationships (for example between two individuals) are only counted as separate relationships if it can be determined that the individuals that are the fillers for the relationships are different to each other.

Cardinality Restrictions

- Example:



- Individual *Matthew* satisfies a minimum cardianlity restriction of 2 along the *worksWith* property if the individuals *Nick* and Hai are distinct individual.

Datatype Properties

- Object properties are properties that have a range that is some class.
- OWL also has the capability to define properties with the range of a simple datatype such as a string or integer.
- OWL comes with a large library of pre-existing datatypes that are mostly imported from XML.

Datatype Properties

- A property with a range that is a simple datatype is known as a datatype property.
- Because datatypes don't have all the power of OWL objects, many of the capabilities for object properties such as having an inverse or being transitive aren't available for datatype properties.